Adopting Open-Source Software Engineering in Computer Science Education

Chang Liu
School of Electrical Engineering and Computer Science
Ohio University, Athens, OH 45701
liuc@ohio.edu

Abstract

Open Source Software Development (OSSD) approaches are a fresh addition to the field of Software Engineering. While it is not yet clear how OSSD fits into traditional Software Engineering theories and frameworks, it is already certain that OSSD as an engineering approach is effective and is here to stay. It is, therefore, important to expose Computer Science students to this new and vibrant software development approach so that they are prepared for the real world.

In this paper, how a class of Ohio University Computer Science students uses an open source project site SourceForge.net to complete their class projects in teams is presented. These projects, which were designed to encourage students to use OSSD to it maximum potential, are described, along with the rational behind their design. Lessons learned from this class are also discussed.

Keywords

Software Engineering, Open Source Software Development, Computer Science Education

1. Introduction

While open source software development (OSSD) approaches were not created and fostered primarily by traditional software engineering (SE) researchers, OSSD did achieve some of the goals sought by SE [14]. In the past few years, OSSD has gained popularity and has reached the corporate world, traditionally not a domain of the usually-grassroots open source projects [6]. Therefore, OSSD is significant enough in the real world that it is important to expose undergraduate computer science students to OSSD as part of their SE education, as Port and Kaiser did in their SE courses [12].

I was the instructor of a dual-listed course "CS442/542 Operating Systems and Computer Architecture I" offered by the School of Electrical Engineering and Computer Science, Ohio University in winter 2003 [10]. Students of this course were either senior undergraduate students or graduate students. Traditionally, this course has a significant project component. Typically, instructors of this course spent about one third of lecture time in dealing with issues related to the projects, such as introduction of tools like *Make*, *Yacc*, *Lex*, and coverage of topics like *X Window* programming and debugging techniques with *gdb*, etc. Students were organized in teams once they become comfortable with smaller personal projects and were ready for larger projects. The success of those larger projects typically depended heavily on not only students' technical skills but also their teamwork and coordination capabilities. For those students who had taken a SE course, this was where they could use their SE knowledge in practice. For others who had yet to take a SE course, this was an excellent exercise to prepare them for the SE course.[1]

This winter quarter, I decided to require all CS442/542 students to adopt open-source software engineering approaches in their projects so that they are exposed to OSSD, which I consider is an important aspect of modern SE. I chose the most popular host of open-source project – SourceForge.net and directed all students to a project site at SourceForge.net that is intended exclusively for Ohio University CS442/542 students [1].

The goals of this class are: 1) to help students learn the principles of modern operating systems through creating their own mimic, incomplete OS-like experimental software systems; 2) to expose students to OSSD so that they learn the basics of OSSD as a software engineering approach, and popular OSSD tools such as CVS; and 3) to facilitate students to do more in one quarter by leveraging on the advantages of OSSD approaches, tools provided by open source project hosting sites, and closer collaboration enabled by those tools.

In the rest of this paper, detailed descriptions of the projects are presented, along with a discussion of how these projects were designed to encourage students to apply OSSD principles such as "release early, release often" [13] and to foster asynchronous collaborations between students. What actually happened in the class, as well as lessons learned from this class, are also presented.

[1] The school of EECS at Ohio University does not specify which course students should take first. As a result, some students take CS442/542 first; others take CS456/556 SE first.

2. The Projects

In CS442/542, students are required to build a simplified, pluggable distributed process management system that allows different resource allocation algorithms to be plugged in. The overall system is divided into five individual projects, some of which are in turn divided in stages. These individual projects are chosen in such a way that students who can better understand and leverage on the strength of OSSD are rewarded.

2.1. The distributed process management system

The distributed process management system that students are required to build in this class is a simulated system. There are three major components in this system: a monitor, a global process manager, and local process managers, as shown in Figure 1.

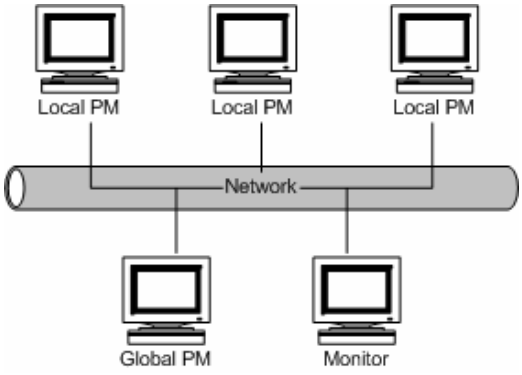

Figure 1. The distributed process management systems.

One instance of the local process manager runs on each host computer in the distributed environment, where process management is to take place. The monitor and the global process manager can run on any host in the network. There can be only one instance of the global process manager in the entire distributed system. There could be, however, more than one instances of the monitor running of different hosts so that current status of the system can be viewed at different locations.

Resources, added to each local process managers via a simple command language, and processes, defined in another simple definition language, are the primary subjects-under-management in this system. Processes typically request and release various amounts of different types of resources throughout their lifetime. The challenge is to find an optimal allocation of limited resources among these processes so that 1) there is no deadlock in the system; 2) when there is a deadlock, it can be detected and the system can recover from the deadlock; and 3) the overheads for deadlock avoidance and recovery from deadlocks are minimal.

Students are free to use any algorithm that they see as fit. For example, they can use the Banker's Algorithm [5] or any other algorithms presented in [15]. They can also come up with their own algorithm.

The global process manager works hand-in-hand with local process mangers and may move portable processes or resources from one host to another host. This opens an entirely new dimension of possibilities for global optimizations and enlarges the playground for students.

2.1.1. The monitor. The monitor is a graphical program that visually displays all actions and events associated with local process managers. The monitor passively accepts data from local process managers and graphically displays the status of all connected local process managers. More specifically, the monitor displays usage and allocations of resources, as well as creation, progress, and termination of processes. Figure 2 shows a partial screenshot of one monitor that was developed by a student.

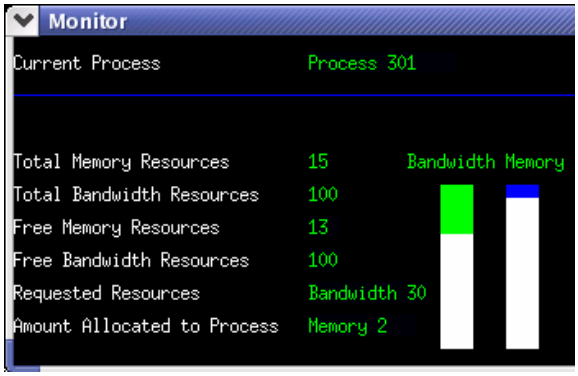

Figure 2. A sample screenshot of a monitor.

```
AddResource pm1 "memory" 15
StartProcess pm1 process2 301
ResourceRequest pm1 301 "memory" 15
AllocateResource pm1 301 "memory" 15
ResourceRequest pm1 301 "memory" 5
AllocateResource pm1 301 "memory" 5
RetrieveResource pm1 301 "memory" 20
StartProcess pm2 process1 301
ProcessEnded pm1 301 0
...
```

Figure 3. Sample messages sent from local PMs to the monitor.

The monitor listens to a TCP port and waits for messages from local process managers. Via these messages, local PMs inform the monitor about any status changes, such as creation and termination of processes, addition or removal of resources, request of resources from processes, allocation of resources to processes or retrieval of resources from processes. The format of these

messages is rather simple. Each message contains an event or action name followed by a few parameters. Several sample messages are listed in Figure 3.

2.1.2. Local process managers. A local process manager is a program that manages all resources and processes on a particular host computer.

Processes are defined in a simple process definition language. This language only offers means to define how a process uses resources. All activities not related to resources are represented by just one command "Compute". An example process definition file is shown in Figure 4.

```
/* process2.def */
ProcessName process2
SetAttribute Portable
RequestResource "memory" 15
Compute 100
ReleaseResource "memory" 5
Compute 100
ReleaseResource "memory"
```

Figure 4. An example process definition file.

Local process managers receive commands either from a file or from the console. These commands typically instruct local process managers how many resources they may use and when to start which process. It is up to the local process managers to decide how to fulfill resource requests from active processes. A few sample commands are shown in Figure 5.

```
SetName pm1
Connect2Monitor panther.cs.ohiou.edu 1542
SetResourceAttribute "bandwidth" Portable
SetResourceAttribute "memory" NotPortable
AddResource "memory" 5
AddResource "bandwidth" 100
StartProcess process1.def
StartProcess process2.def
Pause 20
StartProcess process3.def
```

Figure 5. Sample commands to local PMs.

2.1.3. The global process manager. The global process manager watches whatever happens on all host computers that connect to it. Unlike local process managers, which only have knowledge of resources and processes on their own computers, the global PM has global knowledge of all resources and processes. Therefore, sometimes, the global process manager may be able to offer more optimal ways to allocate resources.

There are two types of actions that a global pm can perform to optimize global resource allocation: 1) relocation of resources from one host to another host; and 2) relocation of processes from one host to another host.

Not all resources can be relocated. Whether or not a resource is portable is defined when the resource is first introduced, as shown in lines 3 and 4 of Figure 5.

In theory, all processes are relocateable across computers of the same architecture, because all computer systems have at least one CPU and some main memory. It may not be beneficial, however, to relocate many processes, because in addition to CPU and main memory, processes typically require accesses to other local resources that are not portable, such as hard drives, displays, and other I/O devices. The cost of accessing these local resources remotely after a process is relocated typically is far higher than the benefits of using remote CPU and memory after the relocation. For certain special types of processes, for example, CPU-bound processes, it may sometimes make sense to relocate them [8].

In this simulated process management system, a definition "SetAttribute Portable" (Figure 4, line 3) is used in a process definition file to indicate that a particular process can be relocated without significant penalty.

Once the global PM decides that it can optimize current resource allocation, it can send commands to local process managers and make adjustments to either resources or processes.

2.2. The design of individual projects

As described in the previous section, the distributed process management system has three major components that interact with each other over the network. There are many different ways of dividing the system into individual projects. In this class, some principles of OSSD [13] were applied in the design of individual projects so that the instructor can: 1) guide students step by step through the development of the whole system, 2) closely follow students' progresses, 3) provide them with prompt feedbacks, 4) facilitate asynchronous collaborations, and 5) encourage motivated students to do more in one quarter.

2.2.1. OSSD Principle No. 1: "Release early, release often". As Raymond stated in [13] (Chapter 4) that "early and frequent releases" effectively helped connect open source projects and their users, and ensure the quality of open source software. To achieve the same goal, I divided the overall distributed process management system into five projects, some of which have dual stages. Students are required to submit compilable code at each stage, as shown in Table 1.

2.2.2. Principle No. 2: "Good open source developers take pride in their code". While some researchers have found that economic incentives and status competition may be reasons that developers contribute to open-source projects, it is a consensus that pride and personal interest

are common reasons behind successful open-source projects [4, 7, 9, 11, 13]. To take advantage of such phenomenon in this class, several motivated students who were interested in GUI design were assigned as GUI-specialists. They continued working on the monitors with GUI while their classmates moved on to develop local process managers (project 3 stage 1). This happened after each student had built a monitor (less networking capability, project 2 stage 2).

Table 1. Individual projects and stages.

Project/Stage	Target
Project 1	Preparation (network programming and debugging)
Project 2	The monitor
Stage 1	Create a hard-coded monitor that can display one pre-defined scenario
Stage 2	Add capability to parse messages from a file.
Stage 3	Add capability to receive msgs from network (GUI-specialists only, in parallel to Prj3-1)
Project 3	The local process manager
Stage 1	Create a simple local PM that reads in commands and prints out actions
Stage 2	Add capability to connect to remote monitors
Project 4	Simple global PM and system integration
Stage 1	Create a "dumb" global PM that does nothing but watches what happens in local PMs.
Stage 2	Add capability to deal with portable resources and processes
Project 5	System optimization and competition of global and local process management algorithms.

2.2.3. OSSD Principle No. 3: Simple, standard protocols. As pointed out by an observer of OSSD, "highly commoditized, simple protocols" is a key to the success of many open source projects, where development takes place in an extremely decentralized, asynchronous fashion [2]. While students in this class met four times a week during lectures, they could hardly find common free time to arrange team meetings outside the class because many students worked many hours a week in addition to several courses they took. As a result, what typically happens in classes that require teamwork, is that students meet shortly once or twice per week in evenings or weekends. They cannot rely on that time to do development. To facilitate teamwork in such an environment that resembles many open source projects, I decided to use simple, standard protocols

These protocols worked extremely well. One most prominent example is the success of monitors. Different monitors developed by different GUI specialists can all work with process managers developed by any students, even though no coordination meeting between them was ever scheduled.

3. Progresses of the Student Projects

Out of 36 students of winter 2003 CS442/542 (including 28 undergraduate students who took CS442 and 8 graduate students who took CS542), only four had SourceForge.net accounts before they were required to open one. All four of them only participated in SourceForge.net projects as "observers". They did not have any development experience with SourceForge.net. Therefore, none of the students had experience with OSSD at SourceForge.net before they took my class. The majority of them had never used tools such as CVS before. While a few students had difficulty submitting code to SourceForge.net for the first time, several of them even incidentally imported unrelated file or overwrote existing file, the majority of the class learned to use CVS remarkable fast.

Students chose to use difficult software packages to build the GUI for their monitors. The most popular choice was X Window programming with C/C++, mainly because this was the only option explained in class.

To test the local process managers, stable, network-enabled monitors are needed. Six students in the class, including three graduate students and three undergraduate students, who were both more interested and stronger in graphic programming, were selected to serve as GUI specialists for the class. They were responsible of each maintaining a monitor, enabling it for network communication, and responding to feature requests and bug reports.

Table 2 shows the distribution of students' choices of GUI packages as well as the distribution of GUI specialists.

As the time of this writing, this class has not concluded yet. However, the complete result of this class will be presented in the workshop.

Table 2. Choice of GUI packages for the monitor.

GUI package	Number of students	GUI Specialists
X Window + C/C++	18	1
Qt + C++	8	2
Java/Swing	4	2
X + Tk lib + C/C++	1	0
GTK 2.0 + C	1	1
X Window + C + a provided simple graphic library	1	0

4. Discussions

Opening up project source code to everybody else in the class is something totally different from traditional practice in Computer Science classes, where plagiarism

was a constant concern and students are often discouraged to look at each other's code. In fact, students were usually warned that tools such as Moss would be used to detect similar code [3].

In this class, it is much easier to copy somebody else's project because now one does not even have to ask that somebody else to give the permission to copy. Interestingly, though, precisely because that now everybody can see everybody else's code and that the code will stay in SourceForge.net repository and remain accessible to the public long after the class is over, the deterrent for plagiarism is in deed much greater. Those few students who would take risk and count on slipping through the grader's eye knows that it is next to impossible to slip through the public's eyes. Whatever mistakes they made will be well documented by the CVS logs on SourceForge.net. As a result, plagiarism was a lesser concern in this class because of OSSD.

Understandably, some students were quite uncomfortable with opening up their source code, many chose to check in their code only shortly before the deadlines. One student was so concerned that he decided to check in .o object files instead of source code (after asking me for permission).

5. Summary

In summary, OSSD is an effective software engineering approach that has its own unique appeals and characteristics. When applied carefully with suitable projects, OSSD approaches can help motivate Computer Science students, facilitate asynchronous collaboration, enable them to learn and do more in class projects, and prevent plagiarism through the strong deterrent of possible public scrutiny.

6. Acknowledgements

The author would like to thank the thirty six Ohio University students of winter 2003 CS442/542 for braving the new world of adopting OSSD in Computer Science education, Dr. Shawn Ostermann of School of EECS for providing course materials from past CS442/542 and answering numerous questions throughout the quarter, and Dr. Karin Sandell of the Center of Teaching Excellence, Ohio University, for encouraging experiments with new teaching methods in classrooms.

References

[1] "CS442/542 Project Site at SourceForge.net," 2003, http://sourceforge.net/projects/ou-cs442-542/.

[2] "Halloween Document," October 1998, http://www.opensource.org/halloween/.

[3] A. Aiken, "Moss: A System for Detecting Software Plagiarism," http://www.cs.berkeley.edu/~aiken/moss.html.

[4] N. Bezroukov, "A Second Look at the Cathedral and the Bazaar," *First Monday*, vol. 4, no. 12, 1999.

[5] E. W. Dijkstra, "Cooperating Sequential Processes," Technological University, Eindhoven, the Netherlands, Technical Report, 1965.

[6] J. Dinkelacker and P. K. Garg, "Corporate Source: Applying Open Source Concepts to a Corporate Environment," The 23rd International Conference on Software Engineering, 1st Workshop on Open Source Software Engineering, Toronto, Canada, May 15, 2001.

[7] FirstMonday, "Interview with Linus Torvalds: What motivates free software developers?," *First Monday*, vol. 3, no. 3, 1998.

[8] I. F. Haddad and E. Paquin, "MOSIX: A Cluster Load-Balancing Solution for Linux," *Linux Journal*, vol. 2001, no. 85es, pp. 6-es, May 1, 2001.

[9] I.-H. Hann, J. Roberts, S. Slaughter, and R. Fielding, "Why Do Developers Contribute to Open Source Projects? First Evidence of Economic Incentives," The 24th International Conference on Software Engineering, 2nd Workshop on Open Source Software Engineering, Orlando, FL, USA2002.

[10] C. Liu, "CS442/542 Winter 2003: Operating Systems and Computer Architecture I," 2003, http://ace.cs.ohiou.edu/~changliu/teaching/Y03W-442-542/.

[11] H. Masum, "Reputation Layers for Open-Source Development," The 23rd International Conference on Software Engineering, 1st Workshop on Open Source Software Engineering, Toronto, Canada2001.

[12] D. Port and G. Kaiser, "Introducing a "Street Fair" Open Source Practice Within Project Based Software Engineering Courses," The 23rd International Conference on Software Engineering, 1st Workshop on Open Source Software Engineering, Toronto, Canada, May 15, 2001.

[13] E. S. Raymond, *The Cathedral & the Bazaar - Musings on Linux and Open Source by an Accidental Revolutionary*: O'Reilly, February 2001. ISBN: 0-596-00108-8.

[14] W. Scacchi, "Is Open Source Software Development Faster, Better, and Cheaper than Software Engineering?," The 24th International Conference on Software Engineering, 2nd Workshop on Open Source Software Engineering, Orlando, FL, USA, May 19-25, 2002.

[15] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, 6th ed: John Wiley & Sons, 2001. ISBN: 0471417432.